Stack Data structure

Stack Class

```java
public class Stack {
    private int size = 100;
    private int top = -1;
    private String[] data = new String[size];
    ...
}
```

Stack Variables

- Can contain ’size’ #Objects.
- This stack is a stack of String Objects. Change ’String’ to the datatype you want to hold in the stack.
- ’top’ points to the current index of the top most value in the stack.
Stack Data structure

Stack Operations

- pop() - Retrieve the top element of the stack.
- push() - Add an element to the top of the stack.
- isEmpty() - Check if the stack is empty.
Stack Data structure

push()

```java
public void push(String element) throws StackOverFlowException {
    if (top == (size - 1)) {
        throw new StackOverFlowException();
    }
    top ++;
    data[top] = element;
}
```
Stack Data structure

pop()

```java
public String pop() {
    if (top == -1) {
        throw new StackEmptyException();
    }
    int curTop = top;
    top--;
    return data[curTop];
}
```
Stack Data structure

isEmpty()

```java
public boolean isEmpty () {
    return top == -1;
}
```

...Experiment with the stack code in course website
Using Stacks

Infix Notations

- $5 + 3$
- $5 + 3 \times 4$
- $(5 + 3) \times 4$

Reverse Polish Notation - RPN

- $5\ 3\ +$
- $5\ 3\ 4\ \times\ +$
- $5\ 3\ +\ 4\ \times$
- Note how priority is encoded without parenthesis
- Also called Postfix
Using Stacks

Infix Notations

- $5 + 3$
- $5 + 3 \times 4$
- $(5 + 3) \times 4$

Polish Notation - Prefix

- $+ 5 3$
- $+ 5 \times 3 4$
- $\times + 5 3 4$
- Again, Note how priority is encoded without parenthesis
RPN - Usage

- Almost all compilers, including Java.
- Very easy to evaluate.
- No need to scan entire input looking for matching parenthesis to identify which operation to do next.
Using Stacks

Evaluate the RPN: $5 \ 3 \ + \ 4 \ *$ using a stack

- Scan input from left to right.
- If operand, push.
- If operator, pop 2 values from the stack, perform the operation and push the result.
Evaluate the RPN: 5 3 + 4 * using a stack

- 5 - operand → push

```
5
```
Using Stacks

Evaluate the RPN: 5 3 + 4 * using a stack

- 3 - operand → push

<table>
<thead>
<tr>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
Using Stacks

Evaluate the RPN: 5 3 + 4 * using a stack

- + , operator → pop two values. i.e pop 3 and 5
- perform the operation. Perform 5 + 3
- push result into stack

8
Using Stacks

Evaluate the RPN: 5 3 + 4 * using a stack

- 4 - operand → push

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
Using Stacks

Evaluate the RPN: 5 3 + 4 * using a stack

- * , operator → pop, perform operation and push result into stack

32
Evaluate the RPN: 5 3 + 4 * using a stack

- We’ve reached the end of input
- If there are no errors in the input, there will be one element in the stack. It will be the result.

32
Expression Trees

In: $5 \times x + 2 \times y$
Post: $5 \times x \times 2 \times y \times +$
Pre: $+ \times 5 \times x \times 2 \times y$

In: $5 \times x + 2 \times y + z \div v$
Post: $5 \times x \times 2 \times y \times + \div z \times v$
Pre: $++ \times 5 \times x \times 2 \times y \times z \div v$
Building expression trees from RPN: 5 x * 2 y * +

- 5 operand. Create tree with single node and push into stack.
Expression Trees

Building expression trees from RPN: $5 \times x + 2 \times y +$

- $'x'$, operand. Create tree with single node and push into stack

```
  x
 /\  \\
5
```
Building expression trees from RPN: 5 x * 2 y * +

- *, operator. Pop two trees from stack, create new tree and push new tree into stack
Building expression trees from RPN: $5 \times 2 \times y +$

- 2, operand. Create tree with single node and push into stack.

- $2 \times 5 \times x$

- And so on, until you have the entire expression tree.
We perform push a, push b, push c, pop, push d, pop pop on empty stack. The current Stack will be?

- a
- a b c d (Stack Top element on the right)
- a b d
- a b
Assignment Questions

How do you perform post order traversal of the tree below?

- postOrder(node.left); postOrder(node.right); print value at node
- print value at node; postOrder(node.left); postOrder(node.right);
- postOrder(node.right); postOrder(node.left); print value at node
Assignments Question

Post order traversal of the tree below will give?

\[ (5 \times 2 \times y + z) \div v + \]

- \[ 5 \times 2 \times y \times + z \div v + \]
- \[ 5 \times 2 \times y \times z + v \div + \]
- \[ 5 \times 2 \times y \times \times z + v \div + \]
- \[ 5 \times 2 \times z + v \div + \]